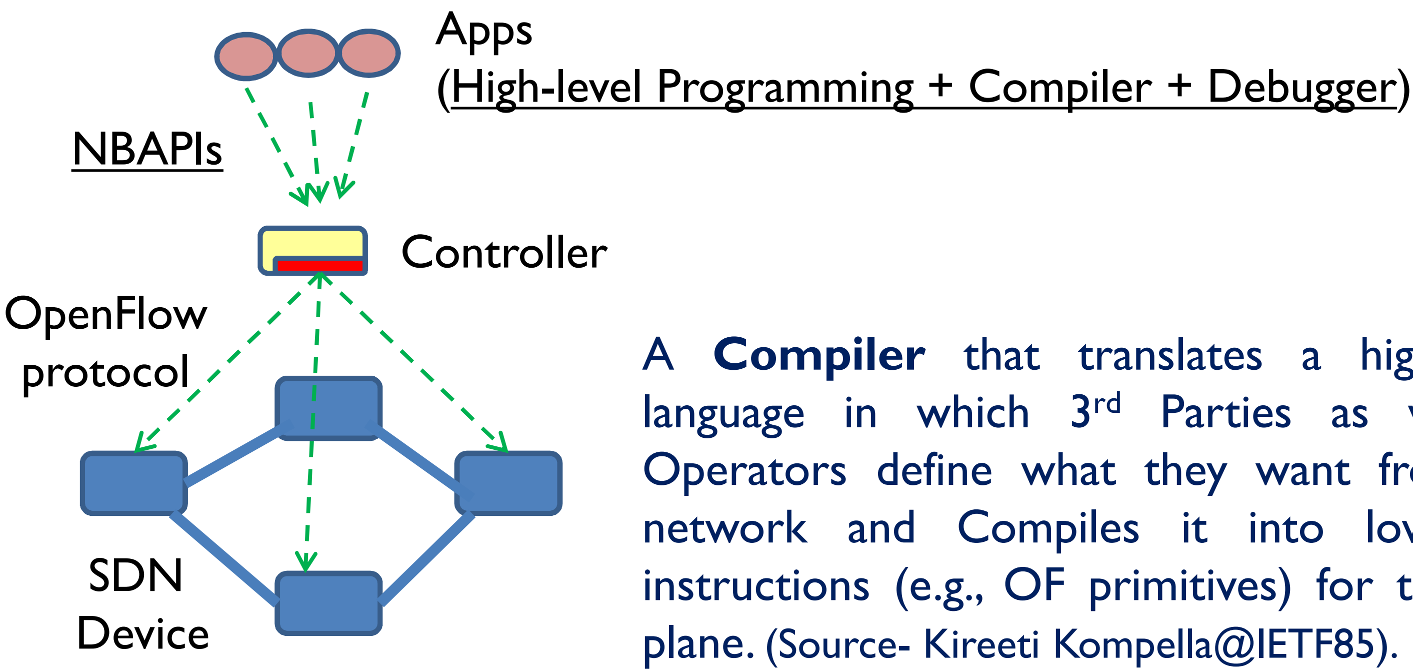
Formal Verification for Software-Defined Networking

Myung-Ki Shin
ETRI
mkshin@etri.re.kr

SDN RG Meeting@IETF 87 – Berlin, Germany

Compiler-based SDN



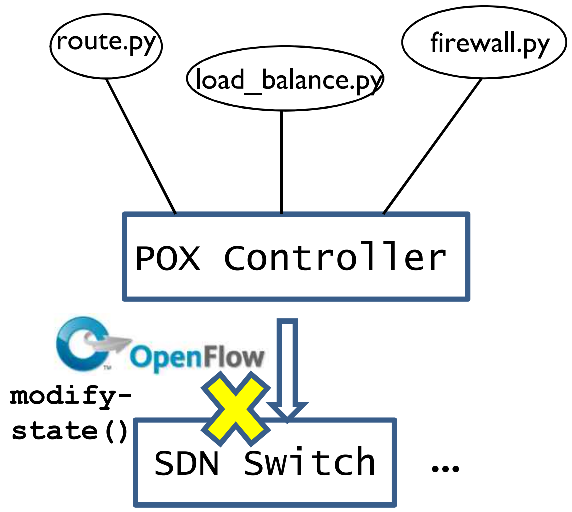
A **Compiler** that translates a high level language in which 3rd Parties as well as Operators define what they want from the network and Compiles it into low level instructions (e.g., OF primitives) for the data plane. (Source- Kireeti Kompella@IETF85).

Why should we verify ?

- To check consistency and safety of network configurations on virtual and physical resources
  - No loops and/or blackholes in the network
  - OF rule consistency between multiple applications on a controller
  - Logically different networks should not interfere with each other (e.g., traffic isolation)
  - New or update configurations conforms to properties of the network and do not break consistency of existing networks (e.g., network updates)

(E.g.) multi-apps on a controller

app1 – route.py / app2 – firewall.py → OF rule conflict

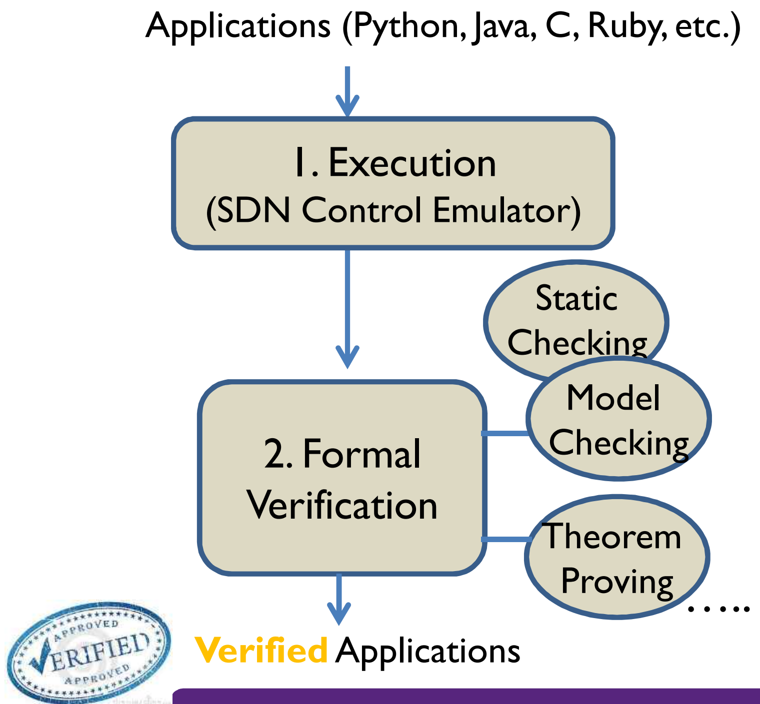


SDN Invariants

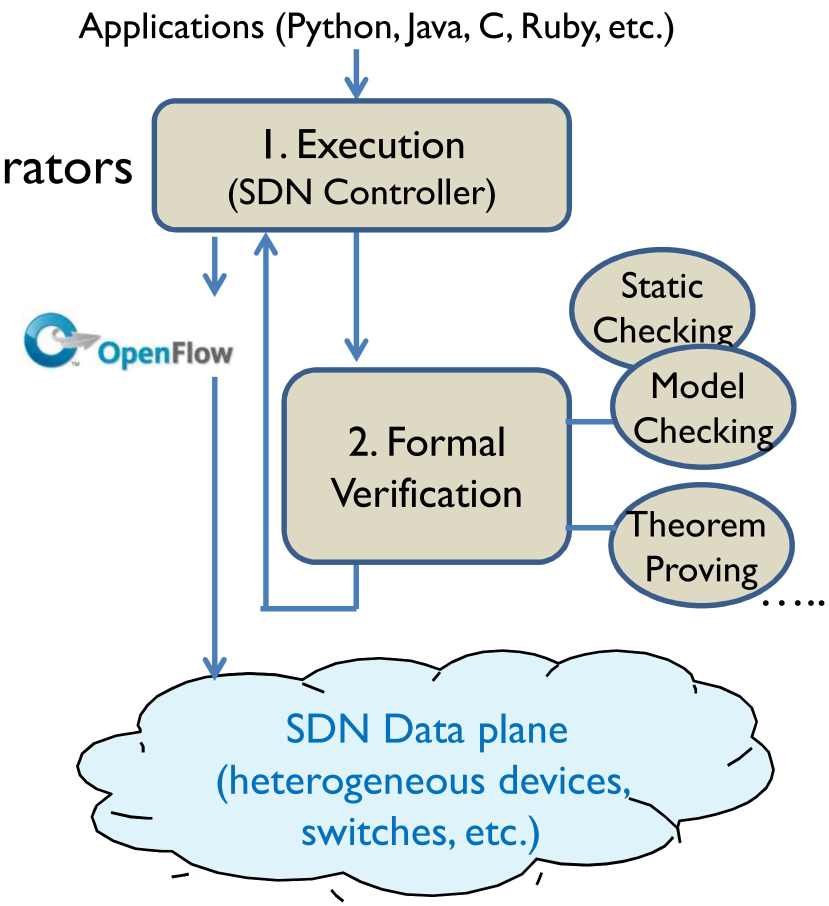
- Basic network properties
  - No loop
  - No blackhole (e.g., packet loss)
- SDN-specific properties
  - OF rule consistency between multiple applications
  - Dynamic info/statistics consistency (e.g., flow, port, QoS, etc.)
  - Consistency with legacy protocols (e.g., STP)

Our Approach:
Formal Verification

- Two Verification Modes
  - Runtime symbolic verification
  - Off-line symbolic verification
- Formal verification is not visible to operators



Off-line symbolic verification

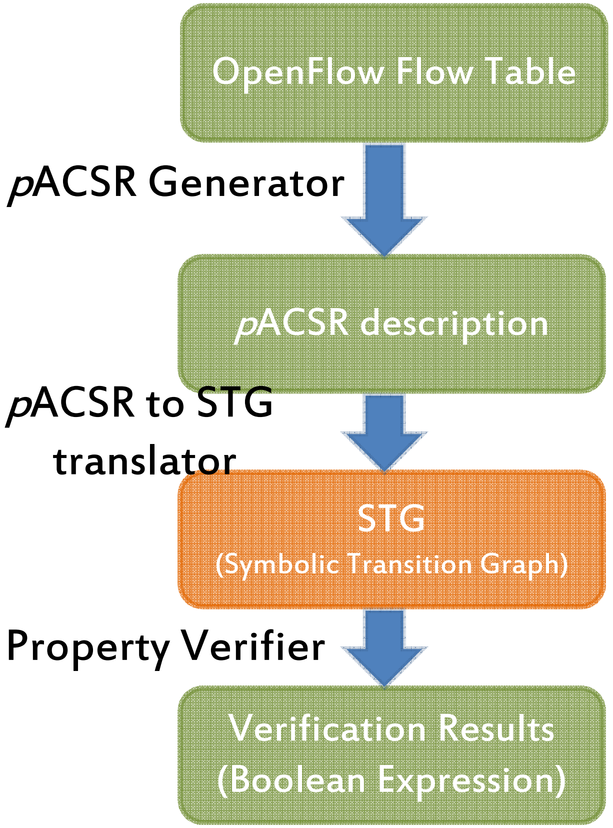


Runtime symbolic verification

What is Formal Verification ?

- Definition from academia
  - A formal description is a specification expressed in a language whose semantics are formally defined, as well as vocabulary and syntax.
  - The need for a formal semantic definition means that the specification language must be based on logic, mathematics, etc., not natural languages.
- Formal verification
  - The act of proving or disproving the correctness of designs or implementations with respect to requirements and properties with which they must satisfy, using the formal methods or techniques

Our Verification Tool Set for SDN (VeriSDN)



- Overall Process
  - Flow table (OpenFlow) is translated into *p*ACSR descriptions
  - *p*ACSR descriptions are fed into VeriSDN Tools
  - In VeriSDN, Symbolic Transition Graph (STG) is generated and various property verification algorithms will be directly applied on STG
  - The result will be boolean expression represented as either BDD or CNF, that show the condition that satisfies the given property

CPS vs. SDN

- ACSR was developed for formal verification of real-time embedded systems and CPS (Cyber Physical Systems).
  - CPS is smart networked systems with embedded sensors, processors and actuators that are designed to sense and interact with the physical world.
  - E.g., Blackout-free electricity generation and distribution; zero net energy buildings and cities; near-zero automotive traffic fatalities and significantly reduced traffic congestion;
    - Guarantee correctness of safety-critical applications for CPS
- In both CPS and SDN,
  - Software is the key (It's the software that determines system/network complexity)
  - There are the same issues on verification of software and its modeling (behaviors).

ACSR (Algebra of Communicating Shared Resources)

$$
\begin{aligned}
P \quad ::= \quad & \mathbf{0} \mid A^u{:}P \mid e.P \mid P + P \mid P \parallel P \mid \\
& P \,\triangle_v\, P \mid P \dagger P \mid [P]_U \mid P \backslash F \mid rec\, X.P \mid X
\end{aligned}
$$

ACSR is a formal language which has notion of Resource, Time, and Priority

- Data Types
  - Basic : Integer, Event Label, Resource Name, Process
  - Composite : Set, Action, Event, Pair
- Operators and Expressions
  - Expressions, Index Definitions, Operand Notation, Precedence and Associativity
  - Integer : Arithmetic, Relational, Boolean, Miscellaneous
  - Sets
  - Process : Prefix, Composition, Context, Miscellaneous
- Commands
  - Miscellaneous, Binding Process Variables, Queries, Process Equivalence Checking, Process Interpretation, Interpreter Commands
- Preprocessor
  - Token Replacement, Macros, File Inclusion, Conditional Compilation ..

*p*ACSR

- *p*ACSR stands for *packet* based ACSR
- *p*ACSR extends ACSR as follows
  - Packets are passed as value (value passing)
  - Parameters are also packets (parameterized process algebra)
  - Predefined predicates and functions are the first class features

predefined predicate

sending packet

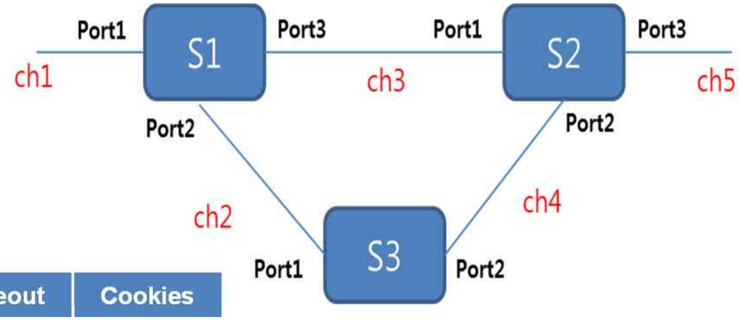
packet

```
P(x) := matchSrcIP(x,sip)→ch!x.nil
S := ch?y.nil
Sys := (P(x) || S)/{ch}
```

parallel composition

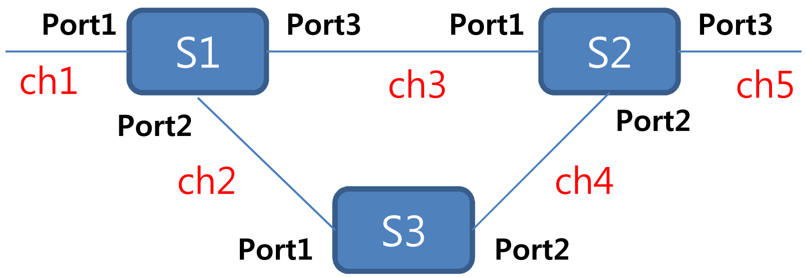
receiving packet

Symbolic Verification Example
(OpenFlow 1.3.1 - Flow Table & Topology)



	Matching	Priority	Counter	Action Set	Timeout	Cookies
S1	in_port1(ch1), ip_src: 10			out_port3(ch3)		
	in_port2(ch2), ip_src: 10			out_port3(ch3)		
	in_port2(ch2), ip_src: 11			out_port3(ch3)		
S2	in_port1(ch3), ip_src: 10			out_port2(ch4)		
	in_port1(ch3), ip_src: 11			drop		
S3	in_port2(ch4), ip_src: 10			out_port1(ch2)		

Flow Table to *p*ACSR



	Matching	Priority	Counter	Action Set	Timeout	Cookies
S1	in_port1(ch1), ip_src: 10			out_port3(ch3)		
	in_port2(ch2), ip_src: 10			out_port3(ch3)		
	in_port2(ch2), ip_src: 11			out_port3(ch3)		
S2	in_port1(ch3), ip_src: 10			out_port2(ch4)		
	in_port1(ch3), ip_src: 11			drop		
S3	in_port2(ch4), ip_src: 10			out_port1(ch2)		

```
S1 := ch1?x.S11(x) + ch2?x.S12(x) + {}:S1
S11(x) := matchSrcIP(x,10)->{}:S13(x)
        + ~matchSrcIP(x,10)->{}:S1 // no rule to match
S12(x) := matchSrcIP(x,10)->{}:S13(x)
        + ~matchSrcIP(x,10)->tau.S14(x)
S13(x) := ch3!x.S1
S14(x) := matchSrcIP(x,11)->{}:S13(x)
        + ~matchSrcIP(x,11)->{}:S1 // no rule to match

S2 := ch3?x.S21(x) + {}:S2
S21(x) := matchSrcIP(x,10)->{}:S23(x)
        + ~matchSrcIP(x,10)->tau.S22(x)
S22(x) := matchSrcIP(x,11)->{}:S2  // drop
        + ~matchSrcIP(x,11)->{}:S2  // no rule to match
S23(x) := ch4!x.S2

S3 := ch4?x.S31(x) + {}:S3
S31(x) := matchSrcIP(x,10)->{}:S32(x)
        + ~matchSrcIP(x,10)->{}:S3 // no rule to match
S32(x) := ch2!x.S3

E := ch1!x.E1
E1 := {}:E1

SDN := (S1 || S2 || S3 || E)/{ch1,ch2,ch3,ch4}
```

*p*ACSR – Operational Semantics

S1 := ch1?x.S11(x) + ch2?x.S12(x) + {}:S1

Switch S1 gets packet from ch1 or ch2 and becomes S11 or S12, respectively. Otherwise, idle one time unit

…

S2 := ch3?x.S21(x) + {}:S2

Switch S2 gets packet through ch3, otherwise, idle.

S21(x) := matchSrcIP(x,10)->{}:S23(x)

Check if source IP of packet ‘x’ is matched with 10

+ ~matchSrcIP(x,10)->tau.S22(x)

Otherwise, try to match other rules

S22(x) := matchSrcIP(x,11)->{}:S2

Check if source IP of packet ‘x’ is matched with 11, and if so, drop it

+ ~matchSrcIP(x,11)->{}:S2

No rule to match, so become S2

S23(x) := ch4!x.S2

Egress packet ‘x’ through ch4

…

E := ch1!x.E1

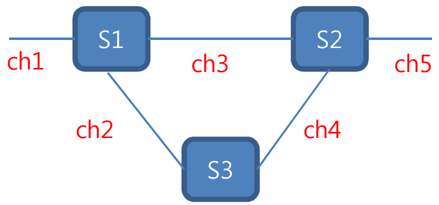
Send packet ‘x’ to ch1

E1 := {}:E1

Idle forever

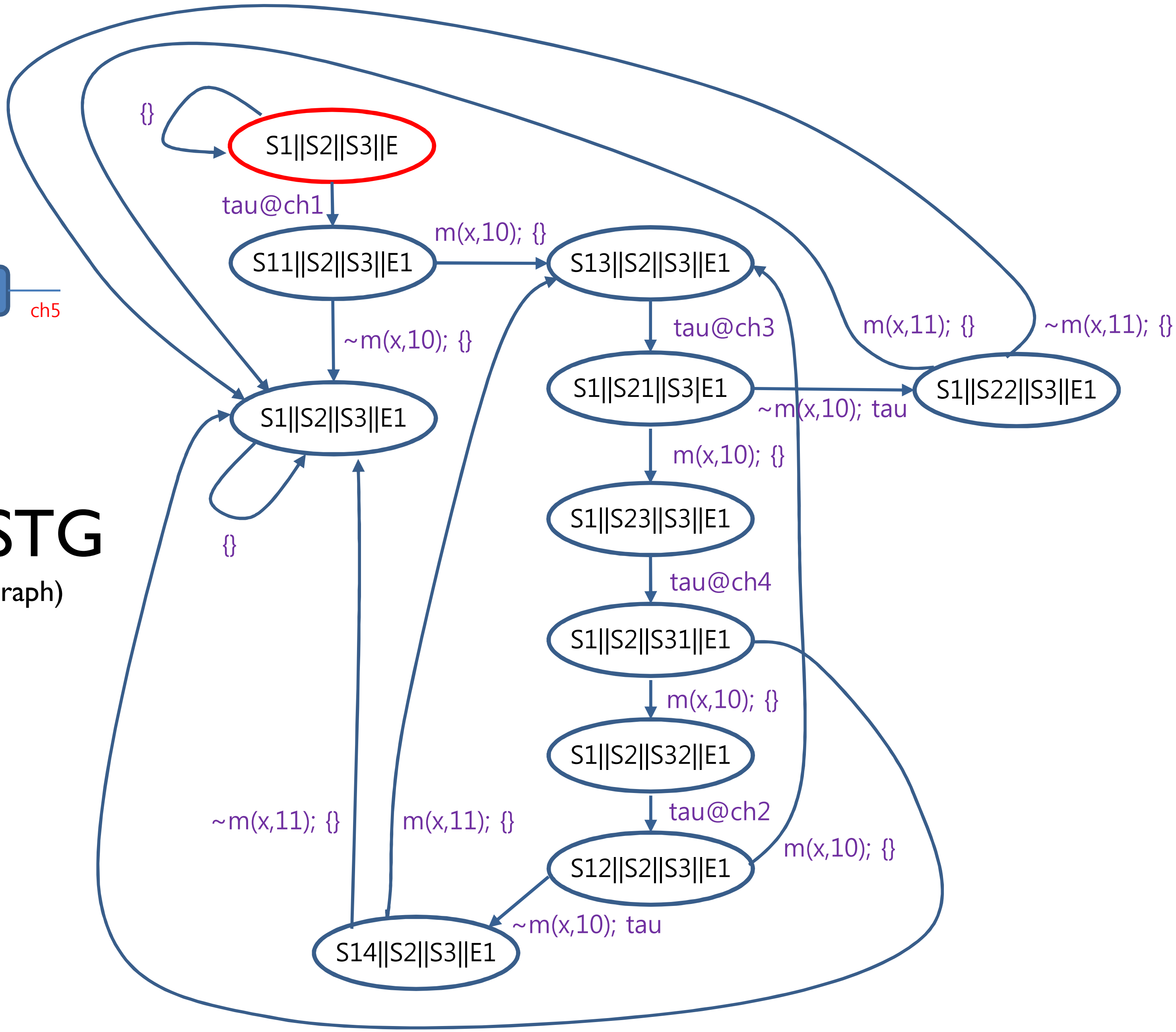
SDN := (S1 || S2 || S3 || E)/{ch1,ch2,ch3,ch4}

Switch S1, S2, S3, and Environment E is running in parallel



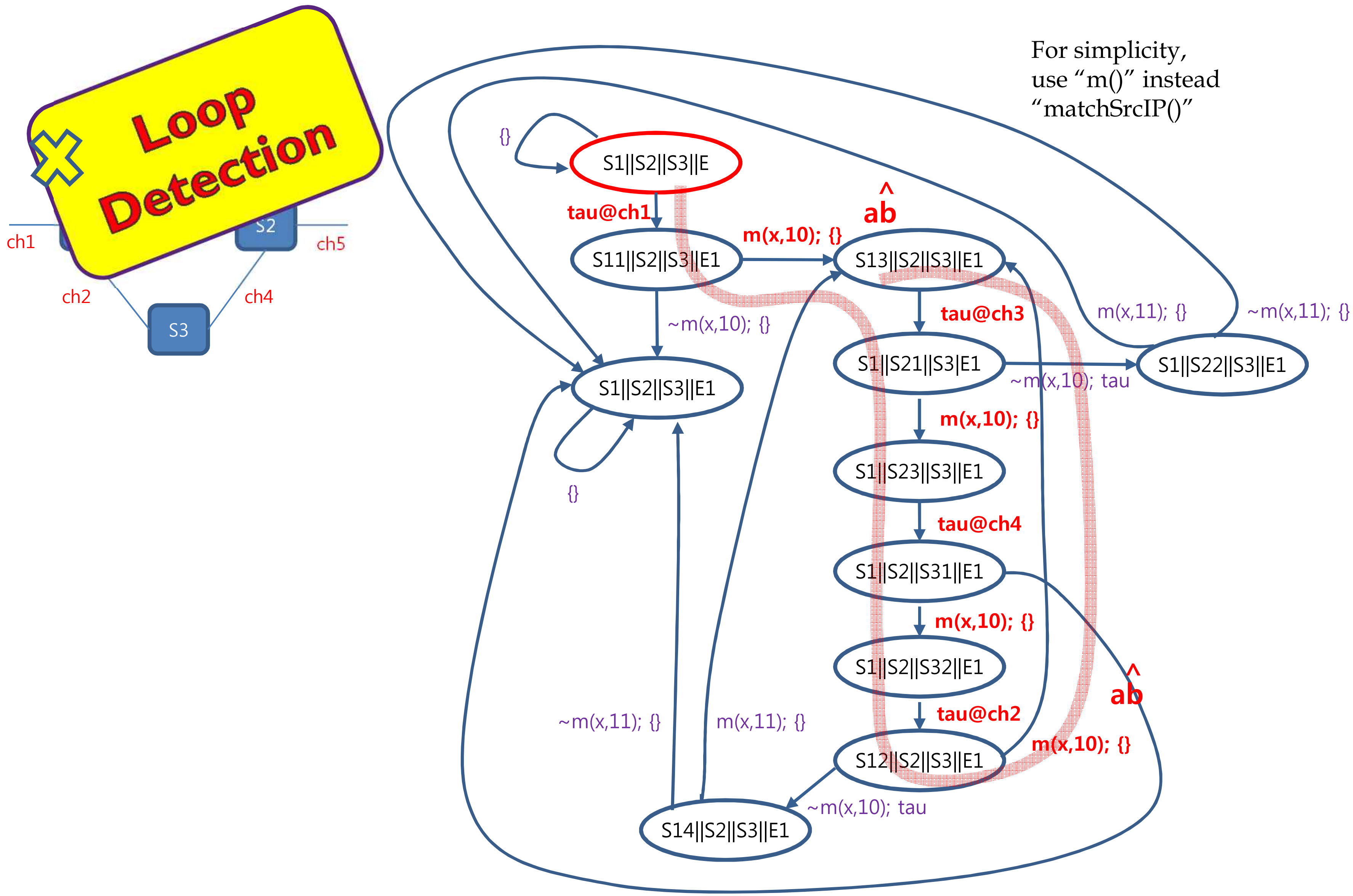
*p*ACSR to STG

(Symbolic Transition Graph)



Symbolic Verification on STG

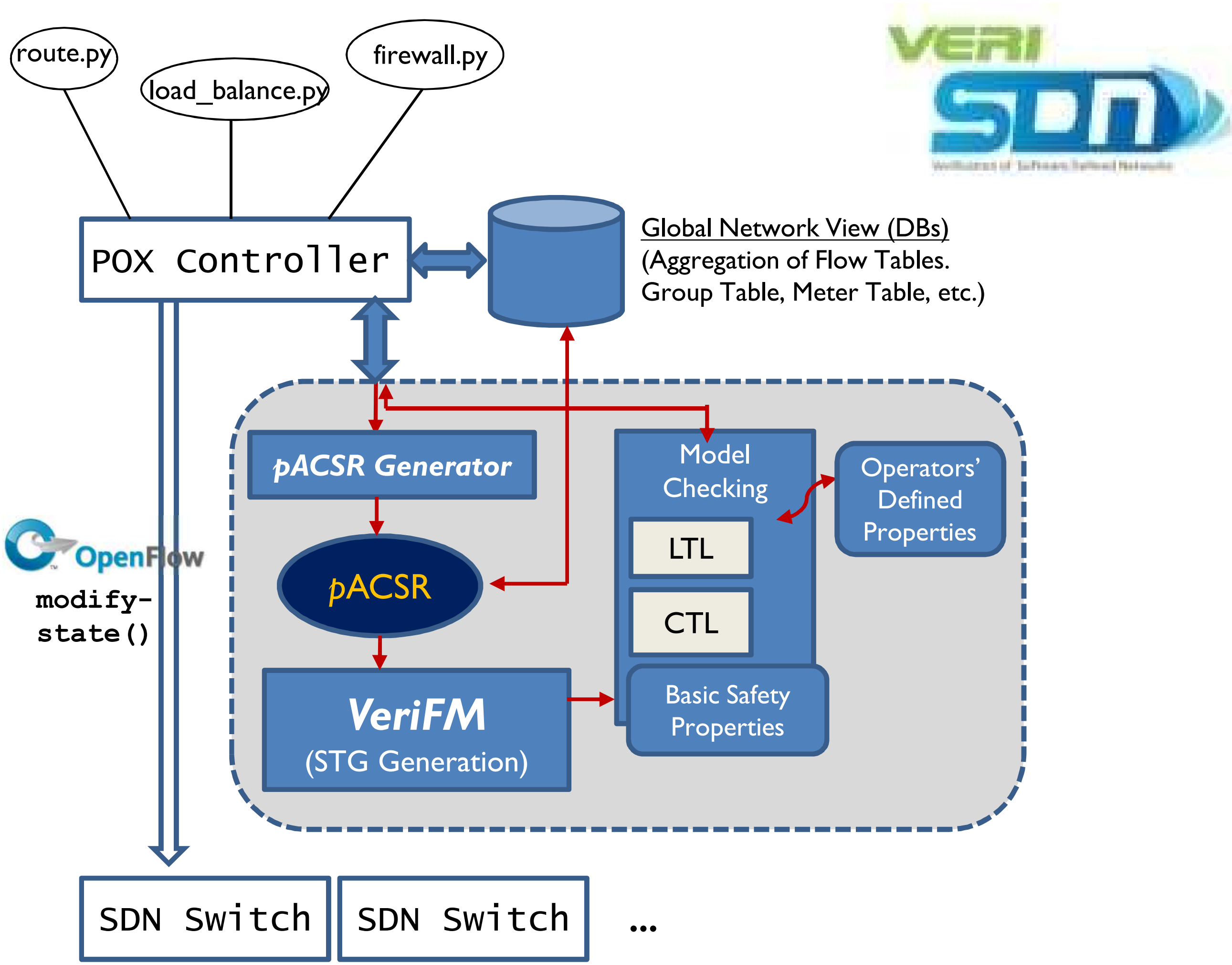
For simplicity,
use “m()” instead
“matchSrcIP()”



VeriSDN: Status

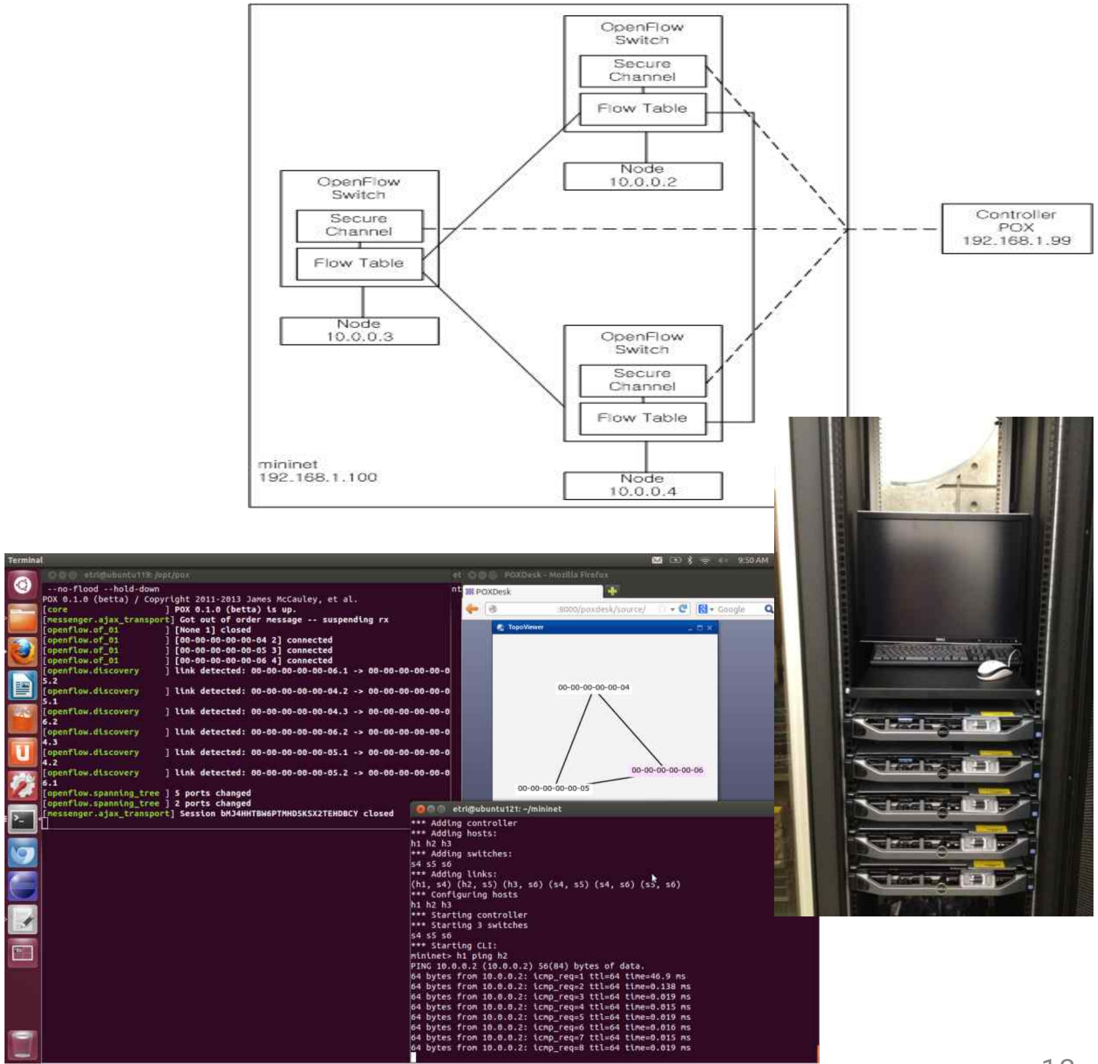
- Wiki - www.veriSDN.net
- Members
  - ETRI, Cemware Co., Ltd., Korea Univ.
- Open source release
  - Initial Release : POX (Python) support tool release(Q4, 2013)
    - C, Javalanguage support (in plan)
  - Target Apps in plan
    - OpenFlow 1.3.x Apps
    - NSC (Network Service Chaining) Validation, Possibly
    - IETF I2RS App (RIB, FIB, QoS, etc), Possibly

Implementation Architecture



Development Environment

- Multi-Apps
  - Routing, Firewall …
- Controller
  - POX (Python)
- VeriFM
  - VERSA (modified)
- Mininet
  - OpenFlow Switch
  - OVS
  - Host
- OpenStack



Discussion and Next Step

- Is "SDNRG" interested in this topic ?
- Investigate relevant works and challenging issues
  - define simple/minimum semantics for SDN abstraction ?
  - Formal description and verification
- Develop a *common* framework document for formal verification of SDN